OBJECT ORIENTED PROGRAMMING THROUGH JAVA

Unit-I

Reference Material

INTRODUCTION

- Java is an object oriented programming language developed by James Gosling & his crew.
- Previously it was called OAK but it's renamed to java during 1995.
- Many versions evolved over the years now the current version is java JDK 8.

KEY ATTRIBUTES OF OOP

- In computer science, computer languages adopt particular programming methodology to write programs.
- For example, C language uses structured programming, Assembly language uses non structured whereas java uses object oriented programming.
- All java programs are written using object oriented programming concepts.
- The oop concepts or key attributes of oop are **encapsulation, inheritance, and polymorphism.**

Encapsulation:

- Encapsulation is the mechanism of hiding the internal details and allowing a simple interface which ensures that the object can be used without having to know its internal details.
- **Example**: A swipe machine encapsulates/hides internal circuitry from all users and provides simple interface for access by every user.
- In java, encapsulation is achieved by binding data and methods in a single entity called class and hiding data behind methods and allowing the interface with the public methods of class.
- The following program demonstrates the concept of encapsulation.

- In the above program data and methods are encapsulated in a class called Customer and data is hidden from user by declaring them to private and interface is allowed to setter and getter methods.

Inheritance:

- Inheritance is a mechanism by which we can define generalized characteristics and behavior and also create specialized ones. The specialized ones automatically inherit all the properties of generalized ones.
- Inheritance can be achieved in java with the extends keyword.
- For example in the above diagram, Vehicle is a generalized class with its own characteristics and behavior, and two-wheeler, four-wheeler classes are specialized ones which inherit all the properties of generalized ones like ID, Name, LicenseNumber.

Polymorphism:

- The ability of an object/operation to behave differently in different situations is called polymorphism.

In the above program the same move() operation is behaving differently in different situations.

SIMPLE PROGRAM

- Operator precedence

```
public class Precedence {
    public static void main(String[] args) {
        System.out.println( 3 + 3 * 2 );
        System.out.println( 3 * 3 - 2 );
        System.out.println( 3 * 3 / 2 );
        System.out.println("--");

        System.out.println( 1 * 1 + 1 * 1);
        System.out.println( 1 + 1 / 1 - 1);
        System.out.println( 3 * 3 / 2 + 2);
        System.out.println("--");

        int x = 1;
        System.out.println( x++ + x++ * --x );

        x = 1;
        System.out.println( x << 1 * 2 >> 1);

        x = 0xf;
        System.out.println( 0xf & 0x5 | 0xa );
    }
}
```

- ***Output:***

```
# java Precedence
9
7
4
--
2
1
6
--
5
2
15
```

- Enter the source code in either editor like notepad or IDE like netbeans or eclipse and save the program name as precedence**.java**
- Compile the program using **javac command** followed by Demo.java in command prompt which generates .class file.
- Execute the program using **java** command followed by Demo which will generates the output.

Description about each part of a program:

- The first line import the classes required for the program. The classes required will be in the subdirectory lang of main directory java. Since the above program requires print() method of system class the package has to be imported into the program.
- The statements written between /* */ are called multiline comments which will be ignored by compiler. It explains the operation of program to anyone reading the program.
- Since java is oop language, every java program is written within a class. So we use **class** keyword followed by class name Demo.
- A class code starts with a { and ends with }. In between { } we can write variables and methods.
- main() is the method which is the starting point for JVM to start execution.
- Since JVM calls the main() method there won't be nothing for the main() to return to JVM. So it is declared as **void** type.

- Since main() method should be available to JVM which is an outside program it should be declared with an access specifier **public**. Otherwise it is not accessible to JVM.
- The keyword **static** allows main() to be executed without creating an object for class Demo. This is necessary because main() is executed by JVM before any objects are made.
- To print something to the console output we use System.out.print() method. System is a class and out is a variable in it. Print() method belongs to PrintStream class.
- When we call **out** variable PrintStream class object will be created internally. So we write System.out.print().
- To print a string to the output we use " "and passed as a parameter to print() method.

Elements of a Java Program

- **Whitespaces:** In Java, whitespace is a space, tab, or newline.We use one whitespace character between each token in java.
- **IDENTIFIERS:** Identifiers are the names given to a variable, class and method. This helps to refer to that item from any place in the program. We can't start an identifier with a digit. Ex: int x; here x is an identifier for a variable.
- **Literals:** A constant value in Java is created by using a *literal* representation of it. For example, here are some literals:

  100 98.6 'X' "This is a test"

- **Comments:** There are three types of comments defined by Java. Single-line, multiline, documentation comment. Documentation comment is used to produce an HTML file that documents the program. The documentation comment begins with a **/**** and ends with a ***/**.
- **Separators:** In Java, there are a few characters that are used as separators. The separators are shown in the following table:

Symbol	Name	Description
( )	Parenthesis	Contains parameters in method call & definition. Contains expressions in control statements. Surrounds cast type.
{ }	Braces	Define block of code for class, method and local scopes. Used to initialize arrays.
[ ]	Brackets	Used to declare arrays.
;	Semicolon	Used to terminate statements.
,	Comma	Separates identifiers in variable declaration. Chains statements together in for statement.
.	Period	Separates package names from sub packages. Separate variable or method from ref. variable.

- **KEYWORDS:** There are 50 reserved keywords currently defined in the Java language. These keywords, combined with the syntax of the operators and separators, form the definition of the Java language. These keywords cannot be used as names for a variable, class, or method.

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used, ** added in 1.2, *** added in 1.4, **** added in 5.0

- In addition to the keywords, java reserves **true, false** and **null**. Const and goto keywords are reserved and meant for future use.

OPERATORS

- An operator is a symbol which tells the compiler to perform a particular operation on operands. Ex: a + b. a and b are operands '+' is an operator meant for addition operation.

- **Categories of operators:**
  - Arithmetic operators
  - Assignment operator
  - Unary operators
  - Relational operators
  - Logical operators or short circuit operators
  - Boolean operators
  - Bitwise operators
  - Ternary or conditional operator
  - Member operator
  - Instanceof operator
  - New operator
  - Cast operator.

- **Arithmetic operators:**

Operator	Operation	example
+	Addition, string concatenation	12+10; String s2="one",s3="two"; String s1=s2+s3;
-	Subtraction	Int x=4,y=3,z; z=y-x;
*	Multiplication	Int x=4,y=3,z; z=y*x;
/	Division	Int x=4,y=3,z; z=y/x;
%	Modulus operator (gives remainder)	Int x=4,y=3,z; z=y%x;

- **Assignment operator:**
  This is used to store a value or variable or an expression into a variable.

Ex: int x=3, y=5, z;
z= x; z=x/3+y;

- **Unary operators:**

Operator	Operation	Example
-	Unary minus(negates given value).	Int x=5; System.out.print(-x));
++	Increments a variable value by 1.	Int x=5; System.out.print(x++);
--	decrements a variable value by 1.	Int x=5; System.out.print(x--);

- Preincrement operator allows incrementation first and then other operations are performed.
  Ex: int x=10;
  System.out.print(++x);// gives result 11 because incrementation first happens & then print operation happens.
- Postincrement operator allows other operations to be performed first and then incrementation happens.
  Ex: int x=10;
  System.out.print(x++);// gives result 10 because print operation happens first incrementation then happens.
- Predecrement operator allows decrementation first and then other operations are performed.
  Ex: int x=10;
  System.out.print(--x);// gives result 9 because decrementation first happens & then print operation happens.
- Postdecrement operator allows other operations to be performed first and then decrementation happens.
  Ex: int x=10;
  System.out.print(x--);// gives result 10 because print operation happens first decrementation then happens.
- **Relational Operators:**
  These are used to perform comparison between elements.

Operator	Operation	example
>(greater than)	Evaluates to true if element greater than other. Otherwise false	System.out.print(2>3);
>=(greater than or equal to)	Evaluates to true if element greater than or equal to other. Otherwise false	System.out.print(2>=3);
<(lesser than)	Evaluates to true if element lesser than other element. Otherwise false.	System.out.print(2<3);
<=(lesser than or equal to)	Evaluates to true if element lesser than or equal to other element .otherwise false.	System.out.print(2<=3);
==(equal to)	Evaluates to true if both elements are equal. otherwise false.	System.out.print(2==3);
!= (not equal to)	Evaluates to true if both elements are equal. otherwise false.	System.out.print(2!=3);

- **Logical Operators:**
  This is used to construct compound conditions.

Operator	**Operation**	**Example**
&&(and operator)	Evaluates to true if each simple condition evaluates to true otherwise false return.	System.out.print(a>b&& b<c);
||(or operator)	Evaluates to true if any one of simple conditions evaluates to true otherwise false return.	System.out.print(a>b||b<c);
!(not operator)	Evaluates to true if condition evaluates to false otherwise true returns.	System.out.print(!(a>b));

- **Boolean operators:**
  These operators act on Boolean variables and produce Boolean type result.

Operators	Operation	Example(a=true b=false)
&(Boolean and operator)	Returns true if both variables true. Otherwise false returns.	System.out.print(a&b)//returns false
|(Boolean or operator)	Returns true if any variable is true. Otherwise false returns.	System.out.print(a|b)//returns true
!(Boolean not operator)	Converts true to false and vice-versa.	System.out.print(!(a|b))//returns false

- **Bitwise operators:**
  These operators act on individual bits of a number and produce the appropriate result.

Operators	Operation	Example x=10, y=11
~(Bitwise Complement Operator)	Gives complement of a given number	(~x)=-11
& (Bitwise and Operator)	Gives 1 if both bits are 1. Otherwise false returns	System.out.print(x&y);//returns 00001010
|(Bitwise or Operator)	Gives 1 if any of the bits are 1. Otherwise false returns.	System.out.print(x|y);//returns 00010100
^(Bitwise xor Operator)	Gives 1 if odd number of 1s present in input. Otherwise false returns.	System.out.print(x^y);//returns 00000001
<<(Bitwise left shift operator)	Shifts bits to the left with the specified number of times.	System.out.print(x<<2);//returns 00101000
>>(Bitwise right shift operator)	Shifts bits to the right with the specified number of times.	System.out.print(x>>2);//returns 00000010
>>>(Bitwise zero fill right shift operator)	Shifts bits to the right with the specified number of times and fill left side empty places with zeroes.	

- **Ternary Operator or Conditional Operator**
  This acts an an alternative for if else statement.
  Syntax: variable =expression1? expression2:expression3;
  Example: max = (a>b)? a :b;
  If exp1 evaluates to true exp2 will be executed. Otherwise exp3 will be executed.
- **Member operator (.)**
  This is used in 3ways.
  - Used when refer to subpackage and class of a package.
    Example: import javax.swing.JOptionPane;
  - Used when refer to variable of a class or object.
    Example: System.out
  - Used when refer to method of a class or object.
    Example: Math.sqrt(23);
- **instanceof Operator**
  This is used to test if an object belongs to a (class or interface) or not.
  Syntax: Boolean variable= object instanceof class;
  Example: Boolean x= custobj instanceof customer;
- **New operator**
  new operator is often used to create objects to classes.
  Syntax: classname obj=new classname();
  Example: customer custobj=new customer();
- **Cast operator**
  Cast operator is used to convert one data type into another data type.
  Syntax: datatype target-var=(target-datatype)variable;
  Example: int x;
  float y=24.5678;
  x=(int)y;
- **Priority of operators**
  In an expression some of the operators will execute first and some operators will execute next.

  To determine which operators execute first priority will be assigned to operators.

  The priority is as follows.

priority	**operators**
1st	() , []
2nd	++ , --
3rd	*, /, %
4th	+, -
5th	Relational operators
6th	Boolean and bitwise operators
7th	Logical operators
8th	Ternary operator
9th	Assignment operator

DATATYPES

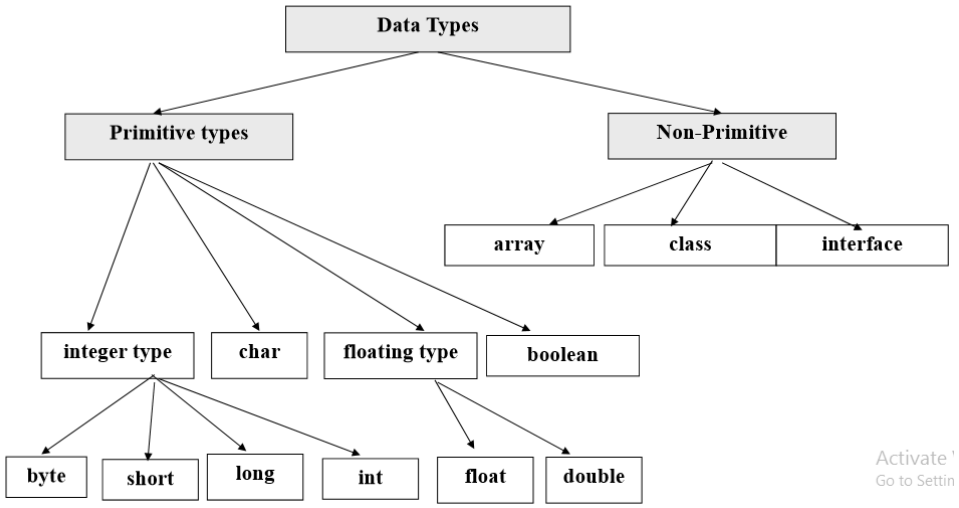
Datatype is an identified type for any data being used in a program.
This is used to determine the following.

- Type of data stored in a variable
- No of bytes it occupies in memory
- Range of data.

In java datatypes are divided into 2 categories.

- Primitive datatypes
- Non primitive datatypes

Under primitive datatypes we have 8 datatypes. Under non primitive we have **class, array, interface.**



The following table shows various datatypes with their details.

Category	Datatype	No.of bytes occupied	Min-max value	Default value	example
Integer type	byte	1byte	(-2^7)- $(+2^7-1)$	0	byte b=10;
	short	2bytes	(-2^{15})- $(+2^{15}-1)$	0	short s=23451;
	int	4bytes	(-2^{31})- $(+2^{31}-1)$	0	int i=2345666;
	long	8bytes	(-2^{63})- $(+2^{63}-1)$	0	long l=342156789;
Floating type	float	4bytes	$(3.4e^{-038})$- $(3.4e^{+038})$	0.0f	Float f=3.45f;
	double	8bytes	$(3.4e^{-308})$- $(3.4e^{+308})$	0.0d	double d=3.2134d;
Textual type	char	2bytes	0- 65535	Null	char ch='x';
Logical type	boolean	1bit	-	false	boolean b=true;

- Float can represent upto 7 digits accurately after decimal point, whereas double can represent upto 15 digits accurately after decimal point.
- char datatype can support 65536 characters including all human language characters which is there in the Unicode system.

VARIABLE

- A variable is an identifier for the data in the program.
- It holds data in a program
- It is named location in memory whose value changes during program execution.
- The type of data stored in the variables can be specified with datatype.
  Syntax: datatype variable-name=value;

Example: int x=23456;

STRINGS

- A string is a set of related characters accessed as a single entity.
- Strings are treated in java as objects of String class.
- String class is defined in java.lang package.
- Once a String object has been created, we cannot change the characters that comprise that string. Each time we need an altered version of an existing string, a new String object is created that contains the modifications. The original string is left unchanged. That's why string objects are **immutable**.
- If a modifiable string is desired, Java provides two options:
  StringBuffer
  StringBuilder.

STRING HANDLING

Java provides various methods and constructors in String class for handling the strings.

1. String s = new String();
   Use: will create an instance of **String with no characters in it.**
2. String(char chars[ ])
   Use: To create a **String initialized by an array** of characters
   example:
   char chars[] = { 'a', 'b', 'c' };
   String s = new String(chars);//output: abc
3. String(char chars[ ], int startIndex, int numChars)
   Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use.
   Use: specify a subrange of a character array as an initializer.
   example:
   char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
   String s = new String(chars, 2, 3);
4. String(String strObj)
   Use: construct a String object that contains the same character sequence as another String object using this constructor
   example:
   String s1="abc";
   String s2=new String(s1);
5. String(byte asciiChars[ ])
   String(byte asciiChars[ ], int startIndex, int numChars)
   Use: constructors that initialize a string when given a byte array.
   Example: byte ascii[] = {65, 66, 67, 68, 69, 70 };
   String s1 = new String(ascii);
   System.out.println(s1);
   String s2 = new String(ascii, 2, 3);
6. String(StringBuffer strBufObj)
   Use: Construct a String from a StringBuffer by using the constructor shown.
   Methods in String class

1. int length()
   Use: returns the length of current String object
   Example:
   String str = new String("Have a nice day");
   System.out.println(str.length());
2. **charAt(int where)**
   Use: returns the character at the specified index.
   Example:

```
char ch;
ch = "abc".charAt(1);
```

3. void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)
   Use: extract more than one character at a time.
   Example:
```
String s = "This is a demo of the getChars method.";
int start = 10;
int end = 14;
char buf[] = new char[end - start];
s.getChars(start, end, buf, 0);
System.out.println(buf);
```
4. String toString()
   Use: returns string of current object String.
   Example:
```
String str = new String("Have  a nice Day");
System.out.println(str.toString());
```
5. byte[ ] getBytes( )
   Use: Performs character-to-byte conversions.
6. char[ ] toCharArray( )
   Use: convert all the characters in a String object into a character array.
7. boolean equals(Object str)
   boolean equalsIgnoreCase(String str)
   use: used to compare two string objects considering case in 1st one and discarding case in 2nd one.
   example
```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Good-bye";
String s4 = "HELLO";
System.out.println(s1 + " equals " + s2 + " -> " +
s1.equals(s2));
System.out.println(s1 + " equals " + s3 + " -> " +
s1.equals(s3));
System.out.println(s1 + " equals " + s4 + " -> " +
s1.equals(s4));
System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
s1.equalsIgnoreCase(s4));
```

8. boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)
   boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)
   Use: compares a specific region inside a string with another specific region in another string.
   if ignoreCase is true, the case of the characters is ignored. Otherwise, case is significant.
9. boolean startsWith(String str)
   boolean endsWith(String str)
   boolean startsWith(String str, int startIndex)
   Example:
```
"Foobar".endsWith("bar")
"Foobar".startsWith("Foo")
```
   Example:
```
"Foobar".startsWith("bar", 3)
```
10. **equals( ) Versus ==**
    Use: the equals( ) method compares the characters inside a String object. The == operator compares two object references to see whether they refer to the same instance.
    Example:

String s1 = "Hello";
String s2 = new String(s1);
System.out.println(s1 + " equals " + s2 + " -> " +
s1.equals(s2));
System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2));

11. int compareTo(String str)
    int compareToIgnoreCase(String str)
    Use: For sorting applications, we need to know which is less than, equal to, or greater than the next. A string is less than another if it comes before the other in dictionary order. A string is greater than another if it comes after the other in dictionary order.

Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.

12. int indexOf(int ch)
    int lastIndexOf(int ch)
    int indexOf(String str)
    int lastIndexOf(String str)
    int indexOf(int ch, int startIndex)
    int lastIndexOf(int ch, int startIndex)
    int indexOf(String str, int startIndex)
    int lastIndexOf(String str, int startIndex)
    Use: used to find the index of the character in a string.
13. String substring(int startIndex)
    String substring(int startIndex, int endIndex)
    Use: used to extract a part of a string object.
14. String concat(String str)
    Use: used to append a string to a invoking string object.
    example:
    String s1 = "one";
    String s2 = s1.concat("two");
15. String replace(char original, char replacement)
    String replace(CharSequence original, CharSequence replacement)
    Use: Replaces all occurrences of one character in the invoking string with another character.
    Example: String s = "Hello".replace('l', 'w');
16. String trim( )
    Use: used to trim extra white spaces for the both sides of string object.
    Example: String s=" head ".trim();
17. static String valueOf(double num)
    static String valueOf(long num)
    Use: The valueOf( ) method converts data from its internal format into a human-readable form.
18. String toLowerCase( )
    String toUpperCase( )
    Use: used to convert the case of the given string object.

StringBuffer class:

String represents fixed-length, immutable character sequences. In contrast, StringBuffer represents growable and writeable character sequences.

Constructors in StringBuffer class

i. StringBuffer( )

Initialize stringbuffer with null characters.

ii. StringBuffer(int size)

Initialize stringbuffer with the specified size of characters.

iii. StringBuffer(String str)

Initialize stringbuffer with the given string.

iv. StringBuffer(CharSequence chars)

Initialize stringbuffer with the given character array.

Methods in StringBuffer class:

1. int length( )
   The current length of a StringBuffer can be found via the length( ) method.
2. int capacity( )
   The total allocated capacity can be found through the capacity( ) method.
   Example:
   ```
   StringBuffer sb = new StringBuffer("Hello");
   System.out.println("buffer = " + sb);//output: Hello
   System.out.println("length = " + sb.length());//output: 5
   System.out.println("capacity = " + sb.capacity()); //output:21
   ```
3. ensureCapacity( )
   If want to preallocate room for a certain number of characters after a StringBuffer has been constructed, you can use ensureCapacity( ) to set the size of the buffer.
4. void setLength(int *len)*
   This method is meant for setting the length for stringbuffer object.
5. char charAt(int *where)*
   This method is used to retrieve a character at a specified index.
6. void setCharAt(int *where, char ch)*
   This method is used to replace a specified position with the given character.
   Example:
   ```
   StringBuffer sb = new StringBuffer("Hello");
   System.out.println("buffer before = " + sb);
   System.out.println("charAt(1) before = " + sb.charAt(1));
   sb.setCharAt(1, 'i');
   sb.setLength(2);
   System.out.println("buffer after = " + sb);
   System.out.println("charAt(1) after = " + sb.charAt(1));
   ```
7. void getChars(int *sourceStart, int sourceEnd, char target[ ],* int *targetStart)*
   This method is used to get the string into the character array from specified position.
8. StringBuffer append(String *str)*
   StringBuffer append(int *num)*
   StringBuffer append(Object *obj)*
   Concatenates the string representation of any other type of data to the end of the invoking StringBuffer object.
9. StringBuffer insert(int *index, String str)*
   StringBuffer insert(int *index, char ch)*
   StringBuffer insert(int *index, Object obj)*
   The insert( ) method inserts one string into another.
   Example:
   ```
   StringBuffer sb = new StringBuffer("I Java!");
   sb.insert(2, "like ");
   System.out.println(sb);
   ```
10. StringBuffer reverse( )
    This is used to reverse a string.
    Example:
    ```
    StringBuffer s = new StringBuffer("abcdef");
    System.out.println(s);
    s.reverse();
    System.out.println(s);
    ```
11. StringBuffer delete(int *startIndex, int endIndex)*
    StringBuffer deleteCharAt(int *loc)*

Used to delete a character at a specified position.
Example:

```
StringBuffer sb = new StringBuffer("This is a test.");
sb.delete(4, 7);
System.out.println("After delete: " + sb);
sb.deleteCharAt(0);
System.out.println("After deleteCharAt: " + sb);
```

12. StringBuffer replace(int *startIndex, int endIndex, String str)*
    We can replace one set of characters with another set inside a StringBuffer object by calling replace( ).
    Example:

```
StringBuffer sb = new StringBuffer("This is a test.");
sb.replace(5, 7, "was");
System.out.println("After replace: " + sb);
```

13. String substring(int *startIndex)*
    String substring(int *startIndex, int endIndex)*
    This is used to obtain a portion of a StringBuffer by calling substring( ).

String Builder Class

- J2SE 5 added a new string class to Java's already powerful string handling capabilities. This
- new class is called StringBuilder.
- It is identical to StringBuffer except for one important difference: it is not synchronized, which means that it is not thread-safe.
- The advantage of StringBuilder is faster performance. However, in cases in which we use multithreading, we must use StringBuffer rather than StringBuilder.